MODULE *DistributedLock*

EXTENDS *Naturals*, *FiniteSets*, *Sequences*, *TLC*

The set of clients
CONSTANT *Clients*

Client states
CONSTANTS *Active*, *Inactive*

Message types
CONSTANT *LockRequest*, *LockResponse*, *TryLockRequest*, *TryLockResponse*, *UnlockRequest*, *UnlockResponse*

An empty constant
CONSTANT *Nil*

The current lock holder
VARIABLE *lock*

The lock queue
VARIABLE *queue*

The current lock *ID*
VARIABLE *id*

$serverVars \triangleq \langle lock, id, queue \rangle$

Client states
VARIABLE *clients*

$clientVars \triangleq \langle clients \rangle$

Client messages
VARIABLE *messages*

Variable
VARIABLE *messageCount*

$messageVars \triangleq \langle messages, messageCount \rangle$

---

The invariant checks that no client can hold more than one lock at a time
$TypeInvariant \triangleq$
$\quad \land \forall\, c \in \text{DOMAIN } clients : Cardinality(clients[c].locks) \in 0 \,..\, 1$

---

Returns a sequence with the head removed
$Pop(q) \triangleq SubSeq(q, 2, Len(q))$

Sends a message on the given client's channel

$Send(m,\ c) \triangleq$
$\quad \land\ messages' = [messages \text{ EXCEPT } ![c] = Append(messages[c],\ m)]$
$\quad \land\ messageCount' = messageCount + 1$

Removes a message from the given client's channel

$Accept(m,\ c) \triangleq$
$\quad \land\ messages' = [messages \text{ EXCEPT } ![c] = Pop(messages[c])]$
$\quad \land\ messageCount' = messageCount + 1$

Removes the last message and appends a message to the given client's channel

$Reply(m,\ c) \triangleq$
$\quad \land\ messages' = [messages \text{ EXCEPT } ![c] = Append(Pop(messages[c]),\ m)]$
$\quad \land\ messageCount' = messageCount + 1$

---

Handles a lock request. If the lock is not currently held by another process, the lock is granted to the client. If the lock is held by a process, the request is added to a queue.

$HandleLockRequest(m,\ c) \triangleq$
$\quad \lor\ \land\ lock = Nil$
$\qquad \land\ lock' = m \mathbin{@@} (\text{"client"} :> c)$
$\qquad \land\ id' = id + 1$
$\qquad \land\ Reply([type \mapsto LockResponse,\ acquired \mapsto \text{TRUE},\ id \mapsto id'],\ c)$
$\qquad \land\ \text{UNCHANGED } \langle queue,\ clientVars \rangle$
$\quad \lor\ \land\ lock \neq Nil$
$\qquad \land\ queue' = Append(queue,\ m \mathbin{@@} (\text{"client"} :> c))$
$\qquad \land\ Accept(m,\ c)$
$\qquad \land\ \text{UNCHANGED } \langle lock,\ id,\ clientVars \rangle$

Handles a *tryLock* request. If the lock is not currently help by another process, the lock is granted to the client. Otherwise, the request is rejected.

$HandleTryLockRequest(m,\ c) \triangleq$
$\quad \lor\ \land\ lock = Nil$
$\qquad \land\ lock' = m \mathbin{@@} (\text{"client"} :> c)$
$\qquad \land\ id' = id + 1$
$\qquad \land\ Reply([type \mapsto LockResponse,\ acquired \mapsto \text{TRUE},\ id \mapsto id'],\ c)$
$\qquad \land\ \text{UNCHANGED } \langle queue,\ clientVars \rangle$
$\quad \lor\ \land\ lock \neq Nil$
$\qquad \land\ Reply([type \mapsto LockResponse,\ acquired \mapsto \text{FALSE}],\ c)$
$\qquad \land\ \text{UNCHANGED } \langle clientVars,\ serverVars \rangle$

Handles an unlock request. If the lock is currently held by the given client, it will be unlocked. If any client's requests are pending in the queue, the next lock request will be removed from the queue and the lock will be granted to the requesting client.

$HandleUnlockRequest(m,\ c) \triangleq$
$\quad \lor\ \land\ lock = Nil$
$\qquad \land\ Accept(m,\ c)$

```
    ∧ UNCHANGED ⟨clientVars, serverVars⟩
∨ ∧ lock ≠ Nil
  ∧ lock.client = c
  ∧ lock.id = m.id
  ∧ ∨ ∧ Len(queue) > 0
      ∧ LET next ≜ Head(queue)
        IN
            ∧ lock' = next
            ∧ id' = id + 1
            ∧ queue' = Pop(queue)
            ∧ Reply([type ↦ LockResponse, acquired ↦ TRUE, id ↦ id'], c)
    ∨ ∧ Len(queue) = 0
      ∧ lock' = Nil
      ∧ Accept(m, c)
      ∧ UNCHANGED ⟨queue, id⟩
∧ UNCHANGED ⟨clientVars⟩
```

---

Returns whether the client associated with the given message is active

```
IsActive(m) ≜ clients[m.client].state = Active
```

Expires a client's session. If the client currently holds the lock, the lock will be released and the lock will be granted to another client if possible. Additionally, pending lock requests from the client will be removed from the queue, and the client's state will be updated to remove any locks.

```
ExpireSession(c) ≜
    ∧ clients[c].state = Active
    ∧ IF lock ≠ Nil ∧ lock.client = c THEN
          LET q ≜ SelectSeq(queue, IsActive)
          IN
              ∨ ∧ Len(q) > 0
                ∧ lock' = Head(q) @@ ("client" :> c)
                ∧ id' = id + 1
                ∧ queue' = Pop(q)
                ∧ Send([type ↦ LockResponse, acquired ↦ TRUE, id ↦ id'], lock'.client)
              ∨ ∧ Len(queue) = 0
                ∧ lock' = Nil
                ∧ queue' = ⟨⟩
                ∧ UNCHANGED ⟨id, messageVars⟩
      ELSE
          ∧ queue' = SelectSeq(queue, IsActive)
          ∧ UNCHANGED ⟨lock, id, messageVars⟩
    ∧ clients' = [clients EXCEPT ![c].state = Inactive,
                                 ![c].locks = {}]
```

---

Sends a lock request to the cluster with a unique *ID* for the client.

$$
\begin{array}{l}
Lock(c) \triangleq \\
\quad \land\ clients[c].state = Active \\
\quad \land\ Send([type \mapsto LockRequest,\ id \mapsto clients[c].next],\ c) \\
\quad \land\ clients' = [clients\ \text{EXCEPT}\ ![c].next = clients[c].next + 1] \\
\quad \land\ \text{UNCHANGED}\ \langle serverVars \rangle
\end{array}
$$

Sends a try lock request to the cluster with a unique *ID* for the client.

$$
\begin{array}{l}
TryLock(c) \triangleq \\
\quad \land\ \ clients[c].state = Active \\
\quad \land\ \ Send([type \mapsto TryLockRequest,\ id \mapsto clients[c].next],\ c) \\
\quad \land\ \ clients' = [clients\ \text{EXCEPT}\ ![c].next = clients[c].next + 1] \\
\quad \land\ \ \text{UNCHANGED}\ \langle serverVars \rangle
\end{array}
$$

Sends an unlock request to the cluster if the client is active and current holds a lock.

$$
\begin{array}{l}
Unlock(c) \triangleq \\
\quad \land\ clients[c].state = Active \\
\quad \land\ Cardinality(clients[c].locks) > 0 \\
\quad \land\ Send([type \mapsto UnlockRequest,\ id \mapsto \text{CHOOSE}\ l \in clients[c].locks : \text{TRUE}],\ c) \\
\quad \land\ clients' = [clients\ \text{EXCEPT}\ ![c].locks = clients[c].locks \setminus \{\text{CHOOSE}\ l \in clients[c].locks : \text{TRUE}\}] \\
\quad \land\ \text{UNCHANGED}\ \langle serverVars \rangle
\end{array}
$$

Handles a lock response from the cluster. If the client's session is expired, the response is ignored. If the lock was acquired successfully, it's added to the client's lock set.

$$
\begin{array}{l}
HandleLockResponse(m,\ c) \triangleq \\
\quad \land\ \lor\ \land\ clients[c].state = Inactive \\
\qquad\quad \land\ \text{UNCHANGED}\ \langle clientVars,\ serverVars \rangle \\
\qquad \lor\ \land\ clients[c].state = Active \\
\qquad\quad \land\ m.acquired \\
\qquad\quad \land\ clients' = [clients\ \text{EXCEPT}\ ![c].locks = clients[c].locks \cup \{m.id\}] \\
\qquad\quad \land\ \text{UNCHANGED}\ \langle serverVars \rangle \\
\qquad \lor\ \land\ clients[c].state = Active \\
\qquad\quad \land\ \neg m.acquired \\
\qquad\quad \land\ \text{UNCHANGED}\ \langle clientVars,\ serverVars \rangle \\
\quad \land\ Accept(m,\ c)
\end{array}
$$

---

Receives a message from/to the given client from the head of the client's message queue.

$$
\begin{array}{l}
Receive(c) \triangleq \\
\quad \land\ \ Len(messages[c]) > 0 \\
\quad \land\ \ \text{LET}\ message \triangleq Head(messages[c]) \\
\qquad \text{IN} \\
\qquad\quad \lor\ \land\ message.type = LockRequest \\
\qquad\qquad\ \land\ HandleLockRequest(message,\ c)
\end{array}
$$

$$
\begin{array}{l}
\lor\ \land\ message.type = LockResponse \\
\quad\ \land\ HandleLockResponse(message,\ c) \\
\lor\ \land\ message.type = TryLockRequest \\
\quad\ \land\ HandleTryLockRequest(message,\ c) \\
\lor\ \land\ message.type = UnlockRequest \\
\quad\ \land\ HandleUnlockRequest(message,\ c)
\end{array}
$$

---

Initial state predicate

$$
\begin{array}{l}
Init\ \triangleq \\
\quad\ \land\ messages = [c \in Clients \mapsto \langle\rangle] \\
\quad\ \land\ messageCount = 0 \\
\quad\ \land\ lock = Nil \\
\quad\ \land\ queue = \langle\rangle \\
\quad\ \land\ id = 0 \\
\quad\ \land\ clients = [c \in Clients \mapsto [state \mapsto Active,\ locks \mapsto \{\},\ next \mapsto 1]]
\end{array}
$$

Next state predicate

$$
\begin{array}{l}
Next\ \triangleq \\
\quad\ \lor\ \exists\, c \in \text{DOMAIN}\ clients : Receive(c) \\
\quad\ \lor\ \exists\, c \in \text{DOMAIN}\ clients : Lock(c) \\
\quad\ \lor\ \exists\, c \in \text{DOMAIN}\ clients : TryLock(c) \\
\quad\ \lor\ \exists\, c \in \text{DOMAIN}\ clients : Unlock(c) \\
\quad\ \lor\ \exists\, c \in \text{DOMAIN}\ clients : ExpireSession(c)
\end{array}
$$

The specification includes the initial state predicate and the next state

$$Spec\ \triangleq\ Init \land \Box[Next]_{\langle serverVars,\ clientVars,\ messageVars\rangle}$$

---

* Modification History
* Last modified Sat *Jan* 27 01:50:12 *PST* 2018 by *jordanhalterman*
* Created *Fri Jan* 26 13:12:01 *PST* 2018 by *jordanhalterman*